5 Web-Trojaner

Im Mai 2000, als das Web so etwa zehn Jahre alt war, richtete Jim Fulton (er gehörte zu den Entwicklern des Zope Application Server [92]) sein Augenmerk auf eine sehr einfache, aber sehr erschreckende Sicherheitsproblematik. Die Zope-Leute nannten das Problem »clientseitige Trojaner« [93] – ein etwas verwirrender Begriff. Das Wort »Trojaner« stammt aus der griechischen Legende über das *trojanische Pferd*. Heute wird der Begriff »trojanisches Pferd« verwendet, um etwas zu beschreiben, das wie ein Geschenk aussieht, in Wirklichkeit aber eine Falle ist. In Zusammenhang mit Computersicherheit bezeichnet der Begriff »ein Programm, das cool aussieht, aber alle Dateien löscht«.

Die von Jim Fulton beschriebene Problematik hat wenig zu tun mit dem Löschen von Dateien und anderem erschreckendem Zeug, das auf der Clientseite passieren kann. Eigentlich hat sie überhaupt nichts mit dem zu tun, was sich die meisten Leute unter »Programmen«, d. h. ausführbaren Dateien, vorstellen. Das Problem, um das es geht, besteht darin, dass es einem Angreifer gelingen könnte, andere Benutzer durch eine E-Mail oder einen Link dazu zu bringen, sich auf eine Website zu klicken, die sie nie im Sinn hatten. Das geht wieder auf die allgemeine Bedeutung des Begriffs »Trojaner« zurück: eine Falle, die sich hinter etwas Harmlosem verbirgt.

In diesem Kapitel wird das Web-Trojaner-Problem ausführlich erklärt und eine serverseitige Lösung vorgeschlagen.

5.1 Beispiele

Abstimmungssite

Das Problem lässt sich am leichtesten anhand von Beispielen beschreiben. Bevor wir uns mit Onlinebanken (ja, dort kann es auch funktionieren) befassen, beginnen wir mit etwas Einfacherem. Um Statistiken zu erfassen, können Benutzer auf einer Abstimmungssite zwischen verschiedenen Alternativen wählen. Die Abstimmungsseite könnte folgenden HTML-Text enthalten:

```
<form action="http://www.voting.example/vote.asp" method="get">
  <input type="radio" name="alt" value="1"/>Foo<br/>
  <input type="radio" name="alt" value="2"/>Bar<br/>
    .
    .
    .
</form>
```

Da im Formular die GET-Methode benutzt wird, besuchen Benutzer beim Abstimmen URLs wie diese, wenn sie das Formular abschicken:

```
http://www.voting.example/vote.asp?alt=2
```

Was hält einen Angreifer davon ab, die obige URL zu kopieren und mit dem Vorwand an viele Leute zu schicken, sie führe zu einem coolen Spiel, Nackedeis, Katastrophenmeldungen, Witzen oder was immer nötig ist, um Leute zum Besuch des Links zu überlisten? Nichts. Folgen mehrere der angesprochenen Personen dem Link, geben sie alle eine Wählerstimme nach dem Dafürhalten des Angreifers ab, und die Statistiken sind letztlich falsch.

Sie haben gegen dieses Beispiel vielleicht mehrere Einwände. Erstens könnten Sie sagen, dass viele Benutzer einem Link wie diesem nicht folgen würden, weil er verdächtig aussieht. Das ist kein Problem. Der Angreifer kann die URL mittels Umleitung vor kundigeren Benutzern verbergen. Statt direkt auf die Abstimmungssite zu zeigen, führt sie zu einer Seite auf seinem eigenen Server, z. B.:

```
http://www.badguy.example/nicejoke.html
```

Selbstverständlich befinden sich im Gegensatz zu den Erwartungen des Benutzers auf der Seite `nicejoke.html` keine Spiele. Stattdessen enthält sie folgenden HTML-Text:

```
<meta http-equiv="refresh"
  content="0,url=http://www.voting.example/vote.asp?alt=2"/>
```

Nachdem der Browser das `meta`-Tag gelesen hat, führt er eine Umlenkung auf die URL zur Abstimmungssite durch, noch ehe der Benutzer merkt, was vor sich geht. Das Gleiche kann mit Hilfe eines Skripts oder `Location`-Headers erreicht werden.

Wie manch einer weiß, ist die Verwendung einer GET-Anfrage in dieser Situation falsch. Und das ist wahrscheinlich Ihr zweiter Einwand. Der RFC 2616 [19] definiert HTTP und legt fest, dass man POST-Anfragen verwenden sollte, wenn die Aktion Seiteneffekte hat. Mit POST ist es unmöglich, die Abstimmungsdetails in einer URL zu kodieren, weil Parameter in POST-Anfragen in der Anfrage selbst verborgen werden. (Einige Web-Anwendungen achten aber nicht auf den

Unterschied zwischen POST und GET, so dass sie GET-Anfragen dort akzeptieren, wo sie ursprünglich POST wollten.) Das hält den Angreifer aber nicht davon ab, sein Ziel zu erreichen. Er verleitet den Browser zum Ansehen eines automatisch abgeschickten Formulars statt zum Besuch der obigen URL:

```
<form name="f" action="http://www.voting.example/vote.asp"
method="post">
  <input type="hidden" name="alt" value="2"/>
</form>
<script>document.f.submit()</script>
```

Das mit `f` bezeichnete Formular enthält ein verborgenes Feld, in dem der `alt`-Parameter bereits auf die Wahl des Angreifers gesetzt ist. Unter dem Formular versteckt sich ein kleines JavaScript-Programm. Das Skript schickt das Formular genauso, als hätte der Benutzer auf einen fiktiven »Abschicken«-Button geklickt. Der Übeltäter kann diesen Code in eine Webseite einfügen und den Benutzer zu deren Besuch überlisten. Bevor sich der Benutzer umsieht, hat er seine Stimme abgegeben.

HTML-formatierte E-Mails

In einem noch raffinierteren Ansatz wird das obige Formular in eine HTML-formatierte E-Mail eingebettet (mehr über HTML-formatierte E-Mails in Anhang C). Mehrere beliebte Mail-Clients können HTML-formatierte Mails anzeigen, und einige führen sogar den JavaScript-Code aus. Die bedauernswerten Benutzer geben also ihre Stimme (bzw. die des Angreifers) allein schon dadurch ab, dass sie ihre Mails lesen, manchmal sogar ohne zu lesen, dann nämlich, wenn das Mailprogramm ankommende Nachrichten automatisch in der Vorschau anzeigt.

Für Fortgeschrittene

Microsoft Outlook nutzt eine Internet-Explorer-Komponente (MSIE), wenn es HTML-formatierte E-Mails wiedergibt. Unter Windows 2000 vom Autor durchgeführte Tests zeigten, dass die in Outlook benutzte MSIE-Instanz alles, auch Session-Cookies, gemeinsam mit einer bereits offenen MSIE-Instanz nutzt. Sie sollten das bis zu unserem Onlinebank-Beispiel weiter unten im Auge behalten, weil es Tür und Tor für einige freche, ferngesteuerte E-Mails öffnet. (Zum Glück führen die Defaults im neueren Windows XP keine Skripte aus, die in HTML-formatierten E-Mails eingebettet sind.)

Authentifizierung

Bisher haben wir eine eher spielerische Abstimmungsanwendung betrachtet. Was uns in Bezug auf Web-Trojaner wirklich das Fürchten lehrt, ist die Tatsache, dass sie auch in Verbindung mit Authentifizierung funktionieren. Da diese Trojaner einen Benutzer oder seinen

Browser zum Besuch einer Site überlisten, geschehen Dinge quasi im Auftrag des Benutzers, so als hätte er sich bereits an der Zielsite angemeldet. (Dieses Thema mag wie Spielkram klingen: Ich *habe* aber gesehen, dass es gemacht wurde. 2002 schickte ein Deutscher eine Nachricht an mehrere Usenet-Gruppen. Die Nachricht enthielt nichts außer einer URL. Jeder, der auf den Link klickte, gab damit ohne es zu wissen eine Stimme für einen bestimmten Schüler auf einer deutschen Schule ab. Dieses Votum wurde in einer Online-Abstimmung gewertet, mit der die Schulverwaltung entscheiden wollte, welcher ihrer Schüler es verdiente, kostenlos nach England zu reisen. Leider weiß ich nicht, wie die Geschichte ausging. Ich hoffe aber, dass die Verwaltung misstrauisch wurde, als sie die abgegebenen Stimmen auszählte, die vermutlich nicht nur die Anzahl der Schüler an der Schule, sondern die gesamte Einwohnerzahl dieser Stadt bei weitem überstieg.)

Angemeldete Benutzer auf gefährliche Sites locken

Angenommen, ein Benutzer ist an seiner Onlinebank angemeldet. Wenn ein Übeltäter sich zum Nachteil dieses Benutzers bereichern möchte, braucht er den Benutzer lediglich dazu überlisten, in seinem Browser diesen HTML-Text anzusehen:

```
<form name="f" action="https://www.bank.example/pay.asp" method="post">
  <input type="hidden" name="from-account" value="1234.56.78901"/>
  <input type="hidden" name="to-account" value="9876.54.32109"/>
  <input type="hidden" name="amount" value="10000.00"/>
</form>
<script>document.f.submit()</script>
```

Da das Opfer bereits an der Bank-Site angemeldet ist und die Bank (wie mehrere mir bekannte Banken) diese Art von Formular akzeptiert, überweist der Benutzer irrtümlich Geld an den Angreifer oder jemanden, den der Angreifer mit den Behörden in Schwierigkeiten bringen will.

Automatische Anmeldung, Domain-Authentifizierung, Single-sign-on

Damit das klappt, muss man das Opfer erreichen, während es an der Zielsite angemeldet ist. Manchmal ist das recht leicht: Wenn der Benutzer die Option »Automatische Anmeldung« (siehe Abschnitt 6.2.4) gewählt hat, ist er sozusagen immer angemeldet. Ebenso, wenn die Zielsite eine Intranetlösung auf der Grundlage von Domain-Authentifizierung ist (wie z. B. NTLM auf Microsoft IIS). Und sofern sich so genannte Single-sign-on-Lösungen wie Microsoft Passport und Sun Liberty Alliance ausbreiten sollten, können Benutzer an mehr Sites angemeldet sein, als ihnen lieb ist. Doch Banken und andere »seriöse« Sites bieten zum Glück/Pech keine automatische Authentifizierung.

Ist der Benutzer nicht ständig angemeldet, muss der Angreifer ihn irgendwie dazu bringen, dass er sich an der Zielsite anmeldet, bevor er ihn zum Ansehen des böswilligen HTML-Texts überlistet. Auf einer

Site, in welcher der Angreifer vielleicht Inhalt einfügt, z. B. ein Diskussionsforum, ist das ziemlich leicht: Füge einfach einen sehr verlockenden Beitrag in das Forum ein mit einem Link, den sich die Leute ansehen sollen. Jeder, der den Beitrag liest, wird garantiert angemeldet (natürlich, wenn die Site eine Anmeldung verlangt, um Beiträge zu lesen).

Social-Engineering

In vielen Fällen kann der Angreifer aber keine Nachrichten in die Zielsite einfügen. Hier muss er zu anderen Maßnahmen greifen. Social-Engineering (siehe Abschnitt 4.1.3) ist die Rettung! Angenommen, die Zielsite ist unsere Bank namens `www.bank.example`, und der Angreifer möchte einen Benutzer nach der Anmeldung zum Ansehen des Überweisungsformulars überlisten. Der Übeltäter könnte dem Opfer eine vermeintlich von der Bank kommende E-Mail senden (mehr über die Fälschung von E-Mail-Absendern in Anhang C):

```
To: opfer
From: sicherheit@beispiel.bank
Subject: Dringend -- bitte sofort lesen!

Sehr geehrter Herr Sorglos,

aufgrund von Unregelmäßigkeiten im Online-Angebot unserer Bank möchten
wir Sie bitten, uns bei der Überprüfung Ihres Kontos zu unterstützen.
Loggen Sie sich dafür möglichst umgehend in Ihr Online-Konto ein. Nach
dem Einloggen gehen Sie bitte auf folgende Web-Adresse und folgen Sie
den dortigen Hinweisen:

  http://www.beispiel.bank@167772161/check.html

Mit freundlichen Grüßen,
Fred Verdächtig, Leiter IT-Sicherheit, Beispiel-Bank AG
```

Die meisten Benutzer würden glauben, die eingebettete URL zeige auf den Webserver der Bank … tut sie aber nicht. Mit dem Klammeraffen (`@`) weist diese URL den Browser an, sich unter einem Benutzernamen von `www.beispiel.bank` mit der Site `167772161` zu verbinden. Die lange Zahl ist die als einzelne 32-Bit-Ganzzahl kodierte IP-Adresse `10.0.0.1` des Angreifers, und `check.html` ist eine Seite, die den HTML-Text enthält, der die Überweisungsanfrage automatisch abschickt. Hat das Opfer angebissen, ist der Angreifer jetzt um einige Euro reicher.

Überweisungen bei Banken

Einige Banken verwenden einen zweistufigen Prozess zur Durchführung von Überweisungen. Im ersten Schritt gibt man alle Details ein – Quell- und Zielkonto, Betrag, Fälligkeitsdatum usw. Alle Informationen werden vorübergehend in der Benutzer-Session gespeichert. Nach dem Einholen der Informationen folgt ein zweiter Schritt, in dem der Benutzer die Richtigkeit der Informationen bestätigen muss. Bitte den-

ken Sie nun nicht, dass diese Banken automatisch sicherer als andere sind, wenn es um Web-Trojaner geht. Je nachdem, wie der Bestätigungsschritt implementiert wurde, können diese Sites genauso wie einstufige Sites ausgetrickst werden. Der Angreifer erstellt eine Webseite mit Frames – einen Frame für jeden Schritt des Zahlungsprozesses:

```
<frameset rows="50%,50%">
  <frame src="http://www.badguy.example/details.html"/>
  <frame src="http://www.badguy.example/confirm.html"/>
</frameset>
```

Die Datei `details.html` enthält ein automatisches Einsendeformular, in dem die Zahlungsdetails eingegeben werden. `confirm.html` beinhaltet ein Skript, das die Bestätigungsseite nach einer angemessenen Verzögerung einsendet, um sicherzustellen, dass sie den Webserver der Bank *nach* der Anfrage mit `details.html` erreicht. Und das war's schon, solange keine Geheimnisse zwischen diesen beiden Formularen mitgeteilt werden müssen.

Ungeachtet der vom Angreifer gewählten Methode kann das Opfer nicht erkennen, dass etwas Verdächtiges vor sich geht, während er die durch die Überweisung erzeugte Seite sieht. Um diese Seite zu verbergen, genügt es, sie in einen winzigen Frame einzubetten, der vom Benutzer nicht entdeckt wird, und irgendwelche fingierten Informationen in einen anderen Frame zu stellen. Eine Alternative besteht darin, den Browser durch Cross-Site-Scripting (Kapitel 4) sofort nach der fertigen Überweisung an eine andere Seite zu dirigieren, falls die Zielsite irgendwo eine lückenhafte HTML-Kodierung aufweist.

5.2 Das Problem

Die meisten Websites, darunter Banken, Shops, Diskussionsforen und was nicht sonst noch alles, sind heute für die eine oder andere Art von Web-Trojaner-Attacken anfällig. Um eine dagegen geschützte Web-Lösung entwerfen zu können, müssen wir das Problem verstehen.

Wenn jemand auf unserer Seite surft, erzeugen wir normalerweise Webseiten, die URLs enthalten und Formulare ausgeben, damit der Benutzer etwas eingeben kann. Web-Trojaner funktionieren, weil es Angreifern gelingt, Opfern diese Angebote zu unterbreiten und so zu tun, als ob wir die Anbieter wären. Um diese Bedrohung zu vermeiden, müssen wir sicherstellen, dass die Aktion eines Benutzers wirklich auf einem Angebot von uns und nicht auf dem eines Dritten basiert.

Referer-Header

Viele Entwickler glauben, dass sich der `Referer`-Header gut zur Überprüfung eignet, ob der Besucher von unserer Site kam. Im All-

gemeinen sollte der `Referer`-Header zwar nicht zu Sicherheitszwecken benutzt werden, weil er von der Clientseite kommt. Im Web-Trojaner-Fall hätte er aber nützlich sein können, wenn da nicht die Tatsache wäre, dass er aufgrund von Vertraulichkeitsgründen ausgefiltert wird. `Referer`-Header fehlen also oft in absolut legitimen Anfragen. Deshalb müssen wir eine Methode finden, die unabhängig davon arbeitet.

Ständige Authentifizierung

In einem möglichen Ansatz muss sich der Benutzer für jede Aktion, die etwas ändert, erneut authentifizieren. Außerdem muss in jedes Formular, das dem Client präsentiert wird, ein Passwortfeld eingefügt werden. Dieser Ansatz wird von vielen Onlinebanken implementiert. Leider ist die häufige Passworteingabe lästig; deshalb versuchen wir unser Glück mit einem anderen Ansatz.

5.3 Eine Lösung

Ein Ticket-System

Um sich vor Web-Trojanern zu schützen, sollten Entwickler ein »Ticket-System« implementieren. Im Mittelpunkt dieses Systems stehen nicht vorhersagbare Zufallszahlen (siehe Abschnitt 6.3), die man als *Tickets* bezeichnet. Das funktioniert so:

- Eine typische Webseite enthält eines oder mehrere Angebote, um eine Aktion auszuführen, die Seiteneffekte hat. Generieren Sie für jedes dieser Angebote eine eindeutige, zufallsgesteuerte Zeichenkette und verknüpfen Sie sie mit dem Angebot.
  Ist das Angebot ein Formular, fügen Sie das Ticket in ein verborgenes Feld ein:

  ```
  <input type="hidden" name="ticket" value="uFnVB5oHiMVMcFTN"/>
  ```

  Ist das Angebot ein Link, hängen Sie das Ticket an die Liste der URL-Parameter an:

  ```
  <a href="vote.jsp?alt=1&ticket=bVGZTa78LVOOZn9n">Yes, I agree</a>
  ```

- Fügen Sie für jedes erzeugte Ticket eine Zeichenkette ein, welche die betreffende Aktion benennt, und speichern Sie die kombinierte Zeichenkette in einem *Ticket-Pool* in der Session des Benutzers, der das Angebot erhält. Wenn die fragliche Aktion z. B. die Löschung eines mit 1234 nummerierten Beitrags bestätigt und das Ticket durch `LZE9QfzQK5mgysK` dargestellt ist, könnte die zu speichernde Zeichenkette `delnote-1234-LZE9QfzQK5mgysK` lauten. Damit haben wir auf der Client- und Serverseite das gleiche Ticket.

- Extrahieren Sie das Ticket jedes Mal, wenn eine Anfrage zur Durchführung einer Aktion vom Benutzer ankommt. Dann fügen Sie den Namen der anstehenden Aktion am Anfang der Zeichenkette ein und suchen im Ticket-Pool der Session nach einem Treffer. Wird ein Treffer gefunden, können Sie davon ausgehen, dass das Angebot von Ihrer Website stammt. Führen Sie in diesem Fall die Aktion durch und entfernen Sie das Ticket aus dem Pool.

Das Ticket-System funktioniert, weil der Angreifer unmöglich die Ticket-Werte, die Sie dem Benutzer geben, erraten kann. Außerdem kann er keine Tickets in den Ticket-Pool des Opfers auf der Serverseite einfügen.

Fallstricke

Trotzdem gibt es leider mehrere Fallstricke. Erstens: Wenn Sie die Tickets in GET-Anfragen einbinden, so dass sie Teil von URLs werden, riskieren Sie, dass Tickets durch `Referer`-Header durchsickern, falls der Benutzer einem Link von Ihrer Site zu anderen Webservern folgt. Das Gleiche kann passieren, wenn Ihre Webseite Bilder oder andere Objekte von fremden Sites beinhaltet. Dies sollte aber kein Problem sein, wenn Sie eindeutige Tickets verwenden und daran denken, am Ende einer Anfrage das Ticket aus dem Pool zu entfernen. GET-Anfragen sollten für Aktionen, die Daten ändern, ohnehin nicht verwendet werden, so dass für solche Anfragen keine Tickets nötig sind.

Das System versagt auch, wenn Ihre Anwendung für Cross-Site-Scripting anfällig ist. Gelingt es einem Angreifer, JavaScript in eine von Ihrem Server generierte Seite einzufügen, kann er Tickets aus der Seite extrahieren. Und es könnte ihm auch gelingen, einen Browser zur Durchführung des zweistufigen Prozesses zu überlisten, d. h. zuerst ein Ticket anzufordern und es dann ohne Benutzerintervention zu verwenden.

Implementierungstipps

Wie Sie sehen, ist zusätzliche Programmierung nötig, um sich vor Web-Trojanern zu schützen. Macht aber nichts: Die Web-Entwicklung wird dadurch zur Herausforderung und weniger langweilig. Nachfolgend einige Implementierungstipps

- Webseiten, die Tickets enthalten, können nicht in Caches zwischengespeichert werden, weil jedes Ticket nur einmal benutzt werden kann. Weisen Sie Browser und Proxys an, keine Seiten mit Tickets in einem Cache (Abschnitt 1.1.3) vorzuhalten, damit jede Seite mit frischen, gültigen Tickets ausgegeben wird.
- Setzen Sie für die Anzahl von Tickets in einem Pool eine Obergrenze. Es werden Tickets übrig bleiben, weil Benutzer nicht unbedingt auf alle unsere Angebote zugreifen. Wird die Grenze erreicht,

entfernen wir das älteste Ticket und fügen ein neues ein. Diese Einschränkung hindert Leute daran, den Speicher absichtlich mit ungebrauchten Tickets zu füllen.

- Session-Timeouts lassen die serverseitigen Ticket-Pools verschwinden. Dadurch erhalten wir möglicherweise legitime Anfragen, für die sich auf dem Server keine passenden Tickets befinden, z. B. von einem Benutzer, der eine Stunde damit verbrachte, haufenweise Details in ein Web-Formular einzugeben. Benutzern passt es meist nicht, wenn wir ihre Eingaben wegwerfen. Wir könnten stattdessen die Webseite mit einem neuen Ticket und dem gesamten angekommenen Text erneut anzeigen. Dazu könnten wir den Benutzer mit einer erklärenden Meldung auffordern, die Durchführung der Aktion zu bestätigen.
- Tickets sind nur für Anfragen erforderlich, die tatsächlich auf dem Server etwas ändern. Ein Benutzer, der z. B. einen Beitrag editieren will, fordert zuerst – normalerweise durch Anklicken eines Links – das Editierformular an. Diese Anfrage ändert nichts, so dass sie nicht mit einem Ticket geschützt werden muss. Ist er mit der Überarbeitung seines Beitrags fertig, sendet er seine Änderungen zurück. Diese Anfrage aktualisiert die serverseitige Datenbank und sollte daher mit einem Ticket geschützt werden.

Sie haben vielleicht festgestellt, dass das Ticket-System eine versehentliche, doppelte Ausführung einer Anfrage (sowohl mit POST als auch GET) verhindert, denn für ein zweites Mal gibt es kein gültiges Ticket.

5.4 Zusammenfassung

Angreifer können ihren Opfern Angebote im Namen einer Zielsite vorsetzen und sie dadurch verleiten, Dinge zu tun, die sie nie beabsichtigt hatten. Solche Angebote können URLs oder automatisch eingesendete Formulare sein, und sie können durch einen beliebigen, verfügbaren Kanal verteilt werden, z. B. über E-Mails und Webseiten außerhalb der Zielsite.

Um eine Website und ihre Benutzer vor diesen Web-Trojanern zu schützen, müssen Web-Entwickler spezielle Sicherheitsmechanismen implementieren, z. B. das in diesem Kapitel vorgeschlagene Ticket-System. Das Ticket-System stellt sicher, dass eine durchgeführte Aktion auf einem Angebot basiert, das von der tatsächlichen Website und nicht von der eines Angreifers stammt.

Als dieses Buch geschrieben wurde, waren Mechanismen zum Schutz vor Web-Trojanern nicht weit verbreitet. Zahlreiche Websites sind also in dieser Hinsicht ein leichtes Opfer.